Análise da Vulnerabilidade XSS em Aplicações *Web*

Jackson Alves Rodrigues[1], Ana Maria Martins Carvalho[2], Lucas Souza Montanheiro[2]

[1]Departamento de TI – Grupo Privé
Rua do Balneário, Quadra 10, Lote 19 – Caldas Novas – GO – Brazil

[2]Instituto Federal Goiano – Campus Morrinhos (IF Goiano)
Caixa Postal 92 – 75.650-000 – Morrinhos – GO – Brazil

alves.j@live.com, ana.carvalho@ifgoiano.edu.br,
lucasmontanheiro10@gmail.com

Abstract. *With the growth and advancement of the Internet and its technologies, there was a great demand for applications that be accessed over the Web. Most of the Web applications currently developed have a range of vulnerabilities, because development is restricted by time and cost. This paper addresses the Cross-site Scripting vulnerability, known as XSS, with the use of the WordPress tool, tests with vulnerability scanners and prevention tips.*

Resumo. *Com o crescimento e avanço da Internet e suas tecnologias, houve uma grande demanda por aplicações que pudessem ser acessadas através da Web. Grande parte das aplicações web desenvolvidas atualmente apresentam uma gama de vulnerabilidades, pelo desenvolvimento ser restringido por tempo e custo. Este artigo aborda a vulnerabilidade Cross-site Scripting, conhecida como XSS, com a utilização da ferramenta WordPress, testes com scanners de vulnerabilidade e dicas de prevenção.*

1. Introdução

A preocupação com segurança em aplicações *web* tem sido debatida por várias empresas, bem como, a segurança dos dados de clientes e parceiros que são administrados por elas. Porém, existem empresas que não se atentam devidamente a segurança, deixando como um acessório ou algo opcional para seus clientes, preocupando-se mais com o tempo e custo de desenvolvimento dos projetos.

A negligência da segurança durante o desenvolvimento de aplicações geram falhas, que podem ser usadas para obter e alterar informações, que na sua grande maioria são sigilosas. Quando indivíduos mal intencionados usam dessas falhas, eles podem roubar informações, interceptar comunicações entre cliente e servidor, parar ou até mesmo destruir o serviço *web*, deixando-o sem acesso ou inutilizado [Luz 2011].

Muitas aplicações não tem segurança empregada, o que é um risco não só para a aplicação, mas também para o cliente que a utiliza. Mas a segurança não é levada em consideração por gestores e desenvolvedores, pois esses não tratam a segurança como um investimento, mas como gasto de tempo e recursos, e caem na ingenuidade de dizer que se a aplicação está funcionando, portanto ela está segura [Luz 2011].

Geralmente é mais barato construir um *software* seguro do que corrigir problemas após a entrega do produto ao cliente, como versão final [OWASP 2012]. Portanto, enquanto a segurança de aplicações *web* não for um requisito primordial no

ciclo de desenvolvimento de aplicações, existirá mais gastos com reparos de falhas e, inclusive, perca de reputação de empresas quando uma falha for explorada por atacantes.

Este artigo contribui para o entendimento de uma das principais vulnerabilidades existentes em aplicações *web*, como um alerta para desenvolvedores e gestores dos riscos que uma aplicação não segura pode causar. Haverá uma abordagem mais detalhada da vulnerabilidade XSS, com o auxílio da plataforma de CMS *Wordpress*, e dicas para desenvolvedores de como se prevenir dessa vulnerabilidade.

2. Principais vulnerabilidades em aplicações *web*

A vulnerabilidade ocorre de acordo com a falha, para que o atacante possa explorar uma vulnerabilidade encontrada, é necessário ter conhecimento de suas causas, consequências e principalmente os passos necessários para a realização de um ataque. [Basso 2010]. Os atacantes podem usar vários caminhos diferentes através da exploração de uma aplicação em busca de vulnerabilidades, forçando a aplicação a receber dados que, em muitos casos, ela não está preparada, causando assim falhas.

Na área de vulnerabilidades em aplicações *web*, em suas causas e riscos, deve-se levar em consideração as contribuições da *Open Web Application Security Project* (OWASP), que é uma organização aberta, sem fins lucrativos que tem por objetivo encontrar e combater as causas e riscos das falhas em aplicações *web*, fazendo com que se torne visível as práticas de segurança em aplicações, para que seja possível tomar melhores decisões sobre os riscos da falta de segurança nessas aplicações [OWASP 2008].

Um dos documentos mais conhecidos da OWASP é o *Top Ten*, que é lançado a cada triênio e classifica as dez principais vulnerabilidades em aplicações *web*, de acordo com sua frequência e seu risco representado [Luz 2011]. O respectivo documento tem como objetivo identificar os riscos mais graves das vulnerabilidades em aplicações *web*, fornecendo informações sobre a probabilidade de ocorrência e impactos técnicos [OWASP 2017a].

3. Vulnerabilidade Cross-Site Scripting (XSS)

O *Cross-Site Scripting* comumente conhecido como XSS, é uma vulnerabilidade que permite que códigos *JavaScript* sejam executados em uma aplicação através do navegador da vítima. De acordo com o *Top Ten* da OWASP [OWASP 2017a] essa vulnerabilidade permite o sequestro de sessão do usuário, inserção de conteúdo hostil, roubo de informações pessoais, desfigurar sites ou redirecionar os usuários para páginas maliciosas.

Falhas de XSS são encontradas com frequência em aplicações *web*, um caso recente aconteceu durante uma auditoria de rotina do *Web Application Firewall* da empresa *Sucuri Labs*, foi descoberto um XSS que afeta o *plug-in Jetpack WordPress*, um dos *plugins* mais conhecido e utilizado por aplicações em *WordPress*. A vulnerabilidade está agregada ao módulo de formulário de contato presente no *plugin* (que é ativado por padrão) em versões inferior ou igual a 3.7. Esta falha pode ser explorada por exemplo: fornecendo um endereço de e-mail mal intencionado no formulário de contato do site. Como não há tratamento adequado na sessão administrativa de “comentários” os invasores podem usar desse *bug* para executar

códigos em *JavaScript*, que vão dar maior liberdade ao invasor, como por exemplo, esconder um *backdoor* para futuras explorações do site. Neste caso não é uma vulnerabilidade XSS refletida, no entanto estamos falando sobre uma vulnerabilidade XSS armazenada, assim possibilitando totalmente ao invasor ter controle arbitrário dentro do servidor *web* do *WordPress*, alterando funções e permissões de usuários, como até mesmo criando campanhas de *phising.* Segundo informações do site *4security* este *plugin* tem uma base de mais de 1 milhão de instalações ativas em servidores do *WordPress*, que podem ser facilmente exploradas [William 2016].

A OWASP classifica as falhas em XSS em três categorias diferentes, são elas: XSS armazenado, XSS refletido e XSS baseado em DOM.

3.1. Ataques XSS Armazenado

O XSS armazenado (*Stored Cross-site Scripting*), ou também conhecido como XSS persistente, recebe este nome porque o código malicioso fica armazenado na aplicação, normalmente no banco de dados, pelo qual sempre que um usuário acessar aquela informação ou recurso o código malicioso será executado junto [UTO *et al.* 2009].

Na Figura 1 temos uma representação de um ataque XSS armazenado, onde, por exemplo, um atacante insere um código malicioso em um campo de comentário e submete à aplicação com a falha, que não faz a validação dos campos de entrada e grava da mesma maneira que foi inserida em seu banco de dados. Quando outro usuário acessar a parte da aplicação afetada, o banco de dados irá recuperar as informações e executar o código malicioso no navegador da vítima, podendo fazer várias operações, tais como sequestro de sessão, redirecionamento de páginas, instalação de *malwares*, desfiguração de uma página HTML, como também de acordo com [Uto *et al.* 2009] criar uma teia de navegadores escravos para executar comandos *JavaScript* arbitrários em um determinado local da *web*.

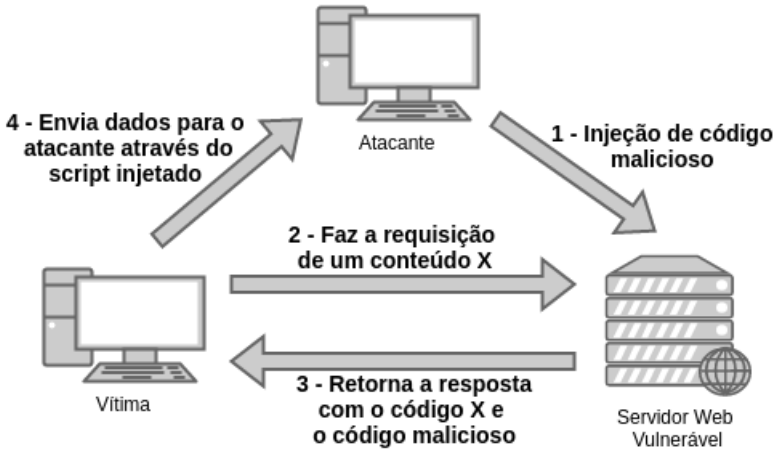

Figura 1. Representação de ataque XSS armazenado

3.2. Ataques XSS Refletido

O XSS refletido (*Reflected Cross-site Scripting*) também conhecido como XSS não persistente, o código malicioso é executado através de uma URL com um código *JavaScript* malicioso no navegador da vítima. Ele é iniciado a partir da ação de um usuário clicar em um *link* contaminado [Oliveira 2012].

Os ataques de XSS refletido costumam-se utilizar de *e-mails* conhecidos como *spam*, que disparam *e-mails* não solicitados para várias pessoas, em sua maioria se passando por entidades financeiras ou serviços solicitando troca de senha, o usuário ao clicar no *link* executa o código malicioso em seu navegador. De acordo com [Uto *et al.* 2009] o ataque pode ser enviado como parte do cabeçalho HTTP, em uma requisição, assim explorando um parâmetro que será exibido sem os devidos tratamentos na página carregada.

Segundo *Testing Guide v3.0* da OWASP [OWASP 2008] o XSS refletido é o tipo mais frequente de ataques XSS em aplicações *web*, pelo fato da facilidade de se alterar parâmetros que são comumente enviados para a aplicação, inserindo códigos *JavaScript* maliciosos e analisando o comportamento da aplicação, a procura da vulnerabilidade.

3.3. Ataques XSS Baseado em DOM

Ataques do tipo *Document Object Model* (DOM) ocorrem devido a uma modificação do "ambiente", ou seja, do navegador da vítima [Luz 2011]. Diferente dos tipos anteriores, armazenado e refletido, a falha ocorre no lado do cliente, fazendo com que apenas no navegador da vítima infectada a aplicação tenha um comportamento diferente do que o servidor enviou à vítima. Esse tipo de ataque costuma ser mais direcionado, e portanto mais difícil de ser encontrado.

Normalmente os vetores de ataques XSS baseados em DOM estão presentes em extensões que são baixadas para dar uma funcionalidade extra aos navegadores, em meio a nova funcionalidade, o atacante insere códigos maliciosos que vão alterar as páginas que a vítima acessar, podendo assim inclusive enviar informações aos atacantes, como por exemplo, senhas ou outras operações.

4. Trabalhos Relacionados

A literatura apresenta esforços e contribuições para tentar mitigar e tratar vulnerabilidades em aplicações *web*, nesta seção será discutido alguns desses trabalhos. [Basso 2010] aborda a eficácia dos *scanners* de vulnerabilidades em aplicações *web*, baseando-se em técnicas de injeção, falhas e modelagem de árvores de ataque. Também verifica-se a relação das falhas por meio do *software* e as vulnerabilidades de segurança.

No trabalho de [Luz 2011] é feito uma análise de vulnerabilidades em *Java Web Applications*. O autor desenvolveu um *software Open Source* para analisar o código fonte de sistemas projetados para *web* ou que foram migrados para a *web*, em busca de falhas de segurança. Neste trabalho encontra-se um estudo das principais vulnerabilidades do documento *Top Ten* da OWASP, anomalias da linguagem *Java*, entre outros. Buscou-se várias ferramentas de *scanner* e teste de vulnerabilidades em aplicações w*eb*, fazendo testes dessas ferramentas em aplicações locais e relacionando os resultados umas com as outras.

[Oliveira 2012] aborda testes de segurança em aplicações *web* segundo a metodologia OWASP. Avaliou-se a segurança em aplicações *web* na categoria de *e-commerce*, por se tratar de operações sensíveis executadas na *Web*. No seu trabalho três aplicações *e-commerce* são testadas: a primeira um *e-commerce* real, a segunda uma aplicação de código aberto e a terceira desenvolvida segundo as recomendações do *Testing Guide* da OWASP [OWASP 2008]. Nos resultados obtidos ele conclui que as três aplicações testadas possuem fraquezas similares.

No trabalho de [Uto *et al.* 2009] são apresentadas as principais vulnerabilidades em aplicações *web* conforme a OWASP, procurando maneiras de detecções de falhas nas aplicações, apresentando modelos de ataques e suas respectivas contramedidas, apresenta também uma lista de controles para evitar a presença dessas vulnerabilidades. Sua contribuição foi expressamente importante, considerando a apresentação de cada vulnerabilidade e seus pontos de exploração.

O trabalho de [Farias 2009] aborda a vulnerabilidade de Injeção SQL nas aplicações *web*, abrangendo as causas e as prevenções. Seu objetivo é expor ao leitor as boas práticas que podem ser utilizadas para mitigar e evitar ataques na aplicação desenvolvida. Apresenta um protótipo desenvolvido pelo mesmo para testar a vulnerabilidade de Injeção SQL e as maneiras de se prevenir desse tipo de falha.

5. Abordagem, experimentos e resultados

Para os experimentos mostrados a seguir, foi desenvolvido uma aplicação *web*, em *WordPress* e propositalmente foi injetado uma falha de XSS, para demonstração nos testes. O objetivo de criar uma página é por motivos éticos, pois não se deve fazer testes em aplicações que não são de sua autoria ou de uma empresa a qual é contratado para fazer testes de penetração. A Lei Federal 12.737, de 30 de novembro de 2012, tipifica como crime, com pena de 3 meses a 1 ano de detenção e multa, para “invasões em dispositivos alheios ou a instalação de vulnerabilidades para obter vantagem ilícita”.

Foi escolhido o *WordPress*, por ser um sistema de gerenciamento de conteúdo, para criação de sites e blogs, de maneira fácil e rápida, que muitas das vezes é utilizada por leigos para desenvolverem aplicações dos mais variados tipos, presentes na Internet. Por ser utilizado, na sua grande maioria, por iniciantes em programação, o *WordPress* e seus *plugins* e temas são recheados de vulnerabilidades dos mais diversos tipos.

Scanners de vulnerabilidade são utilizados por desenvolvedores, testadores e administradores de sistemas, para analisar de forma automática aplicações *Web*, em busca de vulnerabilidades conhecidas por eles. Porém a eficácia da utilização de *scanners* é questionada na literatura, devido ao grande número de falhas não encontradas ou classificadas de maneira errônea [Basso 2010].

5.1. *Wpscan* e sua análise

O primeiro *scanner* a ser abordado é o *Wpscan*, que é uma ferramenta para verificar vulnerabilidades em sites desenvolvidos em *WordPress*, ele é capaz de verificar a existência de vulnerabilidades e informar como explorá-la.

Utilizou-se da ferramenta *Wpscan* para fazer uma varredura geral na aplicação, com essa varredura foi possível identificar a versão da aplicação e uma listagem das possíveis falhas e vulnerabilidades encontradas, além de apresentar um link com maneiras para explorar a vulnerabilidade encontrada. Após o teste foi encontrada uma possível falha de CSRF (*Cross-site Request Forgery*) e foi exibindo links de maneiras para explorá-la, a falha inserida de XSS na aplicação não foi encontrada com a utilização desse *scanner*.

5.2. Acutinex Scanner e sua análise

O *Acutinex* verifica as possíveis falhas em: DDoS, Injeção de SQL, Injeção de PHP, XSS, entre outros. É utilizado tanto por atacantes quanto por desenvolvedores para encontrar vulnerabilidades em aplicações *web*, os desenvolvedores o usam para corrigir as falhas encontradas, já os atacantes aproveitam dessas falhas para invadirem as aplicações a partir das vulnerabilidades listadas. A ferramenta foi utilizada em seu modo de demonstração, já que é uma ferramenta proprietária.

Os testes efetuados com a ferramenta não encontrou a falha XSS injetada, mesmo utilizando a ferramenta para procurar especificamente pela vulnerabilidade XSS, a mesma não conseguiu encontrá-la, encontrando apenas possíveis falhas CSRF.

5.3. Testes manuais

Durante os testes manuais, para provar a existência da falha injetada, foi inserido um comentário em uma página de uma postagem qualquer do *WordPress*, que é a parte da aplicação com a falha injetada, com o código *JavaScript* para que um alerta seja inserido na página, conforme a Figura 2.

Figura 2. Inserindo um código JavaScript em uma entrada com vulnerabilidade *XSS*

A falta de tratamento de dados submetidos na aplicação permitiu que o código *JavaScript* inserido fosse gravado diretamente no banco de dados da aplicação, conforme mostrou o esquema na Figura 1, e agora todos os clientes que acessarem aquela página, visualizarão o comentário, pois o código irá executar automaticamente no navegador da vítima, sendo um ataque do tipo XSS armazenado.

O código injetado na aplicação, para demonstração, apenas executa um alerta no navegador da vítima, com a mensagem “XSS”, porém seria possível inserir códigos maliciosos que poderiam ser capazes de sequestrar sessões, senhas e ou redirecionar para outras páginas maliciosas.

6. Prevenções contra XSS

Para a prevenção de ataques do tipo XSS, deve-se analisar os campos de entradas da aplicação e identificar quais são as entradas mais sensíveis. De acordo com o *Top Ten* da OWASP [OWASP 2017a] a vulnerabilidade ocorre quando as entradas fornecidas por usuários não são filtradas e validadas. Portando XSS dos tipos refletido e armazenado podem ser resolvidos através de validação no lado servidor das aplicações.

A utilização de *White Lists* e *Black Lists* também são aconselhadas para evitar ataques XSS. As listras brancas (*white lists*) são as validações das entradas permitidas nas aplicações, todavia, não são uma proteção apropriadamente completa, pois muitas das aplicações necessitam de caracteres especiais em suas entradas, pode-se também validar o tamanho dos caracteres, e as regras de negócios sobre os dados antes de acessar tal entrada. No caso das listas negras (*black lists*) ao invés de definir o padrão permitido, elas definem o que não é permitido, o que acaba não sendo a solução ideal, pois sempre será necessário atualizar a lista devido a constante quantidade de vetores de ataques que surgem dia após dia. Nesse caso é mais eficiente trabalhar com *white lists*.

Existem também várias bibliotecas de auto sanitização de páginas, que são focadas em segurança para garantir que entradas não sejam mal interpretadas pelos navegadores, possibilitando que os atacantes neutralizem as tentativas de segurança da aplicação. Algumas dessas bibliotecas são a *AntiSamy*, *Java HTML Sanitizer Project*, desenvolvidas pela comunidade OWASP, e também a *Microsoft Anti Cross Site Scripting Library* para as plataformas NET e ASP.NET.

A [W3C 2016], entidade de padronização da Internet, construiu a política de segurança *Content Security Police* (CSP), que está em sua terceira versão, e oferece uma possibilidade de instruir o navegador do cliente a partir de uma localização, ou o tipo de recurso, que está autorizado a ser carregado.

Para os ataques de XSS baseado em DOM a recomendação é sempre baixar extensões confiáveis para navegadores, e conscientizar o usuário do risco desses *downloads*. A OWASP também disponibiliza um artigo para prevenções de falhas baseadas em DOM [OWASP 2017b].

A inserção de um cabeçalho HTTP, chamado de *HttpOnly*, também ajuda na prevenção de XSS, uma vez que ele não permite que um *cookie* seja acessado pelo navegador do cliente, não permitindo que códigos maliciosos acessem os *cookies* salvos pela aplicação.

7. Conclusão

De acordo com os testes efetuados, percebermos que *scanners* nem sempre encontram as vulnerabilidades em aplicações *web*. Para evitar o surgimento de vulnerabilidades devemos usar não apenas *scanners*, é preciso executar vários recursos para detectar uma vulnerabilidade em aplicações *web*, como a revisão de código e deixar a segurança como um requisito durante o ciclo de desenvolvimento da aplicação.

É de extrema importância a conscientização de equipes de desenvolvimento das causas e danos gerados por falhas em aplicações, que por sua vez deixam vulnerabilidades abertas para serem exploradas por usuários mal intencionados, causando grande custo e transtornos para usuários e desenvolvedores da aplicação.

Os experimentos, as prevenções e os modelos de ataques, podem auxiliar iniciantes em segurança de aplicações *web*, ou até mesmo desenvolvedores mais experientes, pois devem se atentar quando estão analisando ou desenvolvendo aplicações, de modo a observar e tratar erros e falhas que podem gerar vulnerabilidades.

Referências

Basso, T. (2010) "Uma abordagem para avaliação da eficácia de scanners de vulnerabilidade em aplicações web". Dissertação de Mestrado, Universidade Estadual de Campinas – Faculdade de Engenharia Elétrica e de Computação. Campinas, São Paulo.

Farias, M. B. (2009) "Injeção de SQL em aplicações Web Causas e Prevenções". Monografia, Instituto de Informática da Universidade Federal do Rio Grande do Sul. Porto Alegre, Rio Grande do Sul.

Lima, W. (2016) "Plugin do WordPress causa falha grave em milhares de sites". Disponível em: <http://4security.com.br/2016/05/31/milhares-de-sites-wordpress-em-risco/>. Acesso em: 10 Abril 2017.

Luz, H. J. F. (2011) "Análise de Vulnerabilidades em Java Web Applications". Trabalho de Conclusão de Curso, Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha. Marília, São Paulo.

Oliveira, T. S. T. (2012) "Testes de Segurança em Aplicações Web Segundo a Metodologia OWASP". Monografia, Departamento de Ciências da Computação, Universidade de Federal de Lavras, Minas Gerais.

OWASP (2008) "*OWASP Testing Guide v3.0*". Disponível em: <https://www.owasp.org/images/5/56/OWASP_Testing_Guide_v3.pdf>. Acesso em: 10 Março 2017.

OWASP (2012) "*OWASP Secure Coding Practices Quick Reference Guide*". Versão 1.3 ptBr Disponível em: <https://www.owasp.org/images/b/b3/OWASP_SCP_v1.3_pt-BR.pdf>. Acesso em: 10 Março 2017.

OWASP (2017a) "*OWASP Top 10 2017 – The Ten Most Critical Web Application Security Risk*". Disponível em: <https://github.com/OWASP/Top10/raw/master/2017/OWASP%20Top%2010%20-%202017%20RC1-English.pdf>. Acesso em: 10 Março 2017.

OWASP (2017b) "*DOM based XSS Prevention Cheat Sheet – Last revision: 08/05/2017*". Disponível em: <https://www.owasp.org/index.php/DOM_based_XSS_Prevention_Cheat_Sheet>. Acessado em: 11 Abril 2017.

Uto, N.; Melo, S. P. (2009) "Vulnerabilidade em Aplicações Web e Mecanismos de Proteção", Cap. 6, IX Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais. Minicursos SBSeg, São Paulo.

W3C (2016) "*Content Security Policy – Level 3*". Disponível em: <https://www.w3.org/TR/2016/WD-CSP3-20160913/>. Acesso em: 10 Março 2017.